RAJIV GANDHI INSTITUTE OF PETROLEUM TECHNOLOGY, JAIS, AMETHI

Department of Computer Science and Engineering

B.Tech. 3rd Year

Software Engineering (CS331)

By

Dr. Kalka Dubey

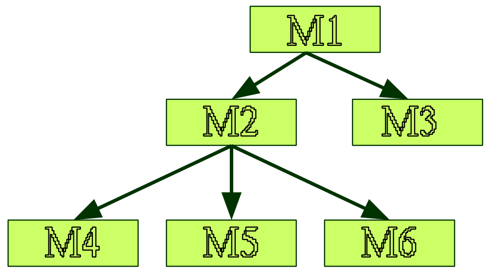
UNIT- 4 (System Design)

- Objective
- Principles
- Module level concepts
- Coupling and Cohesion
- Methodology- structured and object oriented
- Design specification and verification

Introduction

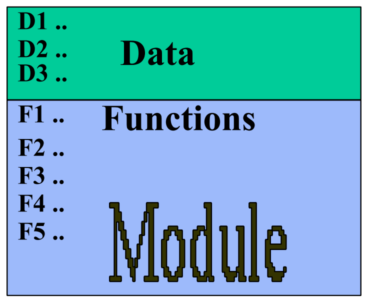
- Design phase **transforms** SRS document:
  - To a form easily implementable in some programming language.

Module Structure



Module Structure

- A module consists of:
  - Several functions
  - Associated data structures.



Items Designed During Design Phase

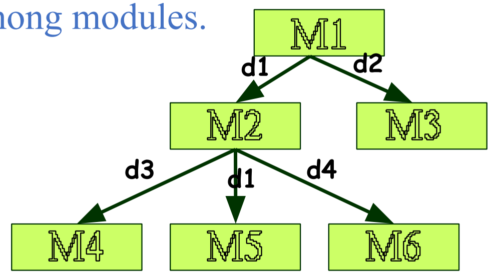
- **Module structure**
- **Control relationship among the modules**
  - call relationship or invocation relationship
- **Interface among different modules**
  - Data items exchanged among different modules,
- **Data structures of individual modules**
- **Algorithms for individual modules**

Software designs

- Design activities are usually classified into two stages:
  - **Preliminary (High-level) design.**
  - **Detailed design.**

High-Level Design

- Identify:
  - **Modules**
  - Control **relationships** among modules
  - **Interfaces** among modules.



High-Level Design

- The **outcome** of high-level design:
  - **Program structure (or software architecture).**

High-Level Design

- Several notations are available to represent high-level design:
    - Usually, a **tree-like diagram** called **structure chart** is used.
    - Other notations:
        - **Jackson diagram** or **Warnier-Orr** diagram can also be used.

Detailed Design

- For each module, design:
  - **Data structure**
  - **Algorithms**
- Outcome of detailed design:
  - **Module specification.**

A Classification of Design Methodologies

- **Procedural** or **Structure** oriented (Function-oriented)
- Object-oriented
- More recent:
  - Aspect-oriented
  - Component-based (Client-Server)

Good and Bad Designs

- There is **no unique** way to design a system.
- Even using the **same design** methodology:
  - **Different designers** can arrive at very **different design solutions.**
- Need to **distinguish between good and bad designs**.

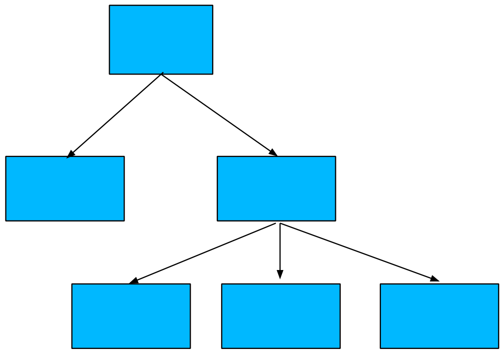
Good Design

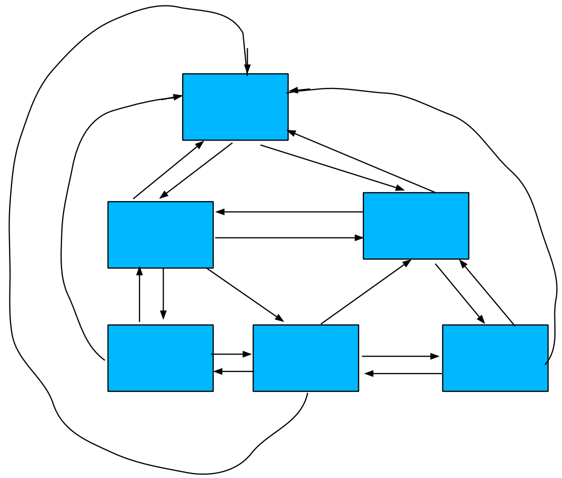
- It should implement **all functionalities** of the system correctly?
- It should be easily u**nderstandable**.
- It should be **efficient**.
- Should be easily **amenable** to change,
  - i.e. easily maintainable.

Good Design

- **Understandability** of a design is a **major one**:
  - It determines the **goodness** of design:
  - A design that is **easy to understand**:
    - Also easy to maintain and change.

Layered Design





Layered Design

- Arrangement of modules in a hierarchy means:
  - **Low fan-out (**The fan-out of a module is the number of its immediately subordinate modules**)**
  - **Control abstraction**

Modularity

- Modularity is a fundamental attributes of any good design.
  - **Decomposition** of a problem cleanly into modules.
  - Modules are almost **independent** of each other.
  - **Divide and conquer** principle.

Modularity

- Modules should have:
  - **High Cohesion**
  - **Low Coupling**.

Advantages of Functional Independence

1. Better **Understandability** and good design.

2. The complexity of design is reduced.

3. Different modules can easily be understood in isolation.

Advantages of Functional Independence

4. Functional independence reduces error propagation.

- Degree of **interaction between modules** is low.
- An **error existing in one module** does **not directly** affect other modules.

5. Reuse of modules is possible.

Cohesion and Coupling

- Cohesion is a measure of:
  - **functional strength** of a module.
  - A cohesive module performs a **single task** or function.
- Coupling between two modules:
  - A measure of the degree of **interdependence** or **interaction** between the two modules.

Cohesion and Coupling

- A module having **high cohesion** and **low coupling**:
  - **Functionally independent** of other modules:
    - A functionally independent module **has minimal interaction** with other modules.

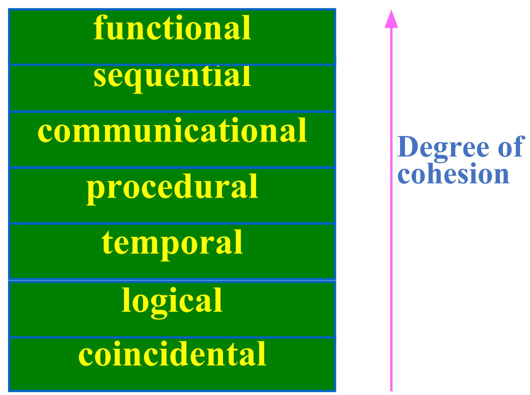
Cohesion

- Unfortunately, there are no ways:

  - To **quantitatively** measure the **degree of cohesion and coupling**.

  - **Classification** of different kinds of cohesion and coupling:
    - Can give us some idea regarding the **degree of cohesiveness of a module**.

Classification of Cohesiveness



Coincidental Cohesion

- The **module performs a set of tasks**:
  - Which relate to **each other very loosely**, if at all.
    - The module contains a **random collection** of functions.
    - Functions have been put in the module out of **pure coincidence** without any **thought or design**.

Logical Cohesion

- All elements of the module perform **similar operations**:
  - e.g. error handling, data input, data output, etc.
- An example of logical cohesion:
  - A set of **print functions** to generate an output report arranged into a single module.

Temporal Cohesion

- The module contains tasks that are **related by the fact**:
  - All the tasks must be executed in the same **time span**.
- <u>Example:</u>
  - The set of functions responsible for
    - initialization
    - start-up
    - shut-down
    - etc.

Procedural Cohesion

- The **set of functions** of the module:
  - **All part of a procedure** (algorithm)
  - Certain sequences of steps have to be carried out in a **certain order** to **achieve** an objective.
    - e.g. the **algorithm for decoding a message**.

Communicational Cohesion

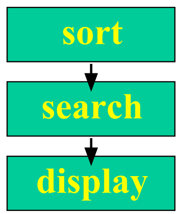
- All **functions of the module**:
  - **Reference** or **update** the same data structure,

- Example:
  - The set of functions is defined on an **array or a stack**.

Sequential Cohesion

- **Elements of a module** form **different parts of a sequence.**

- Output from one element of the sequence is input to the next.
  - Example:



Functional Cohesion

- **Different elements** of a module **cooperate**
  - To achieve a **single function**,
  - e.g. managing an employee's payroll.
- When a **module displays functional cohesion**,
  - Can describe the function using a single sentence.

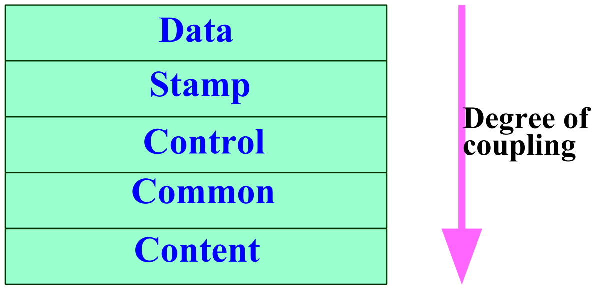
Coupling

- Coupling indicates:
  - How closely **two modules interact** or how **interdependent** they are.
  - The **degree of coupling** between two modules depends on their **interface complexity**.

Coupling

- There are **no ways to precisely way to** determine the coupling between two modules:
  - Classification of different coupling types will help us **approximate and estimate** the degree of coupling between two modules.
- **Five types of coupling** can exist between any two modules.

Classes of coupling



Data coupling

- Two modules are data coupled
    - If they **communicate via a parameter**:
        - an elementary data item.
        - e.g an integer, a float, a character, etc.
    - The data item should be **problem-related**:
        - Not used for **control purposes**.

Stamp Coupling

- Two modules are **stamp coupled**
  - If they communicate via a **composite data item**
    - such as a **record** in PASCAL
    - A **structure** in C.

Control Coupling

- Data from one module is used to direct:
  - **Order of instruction execution in another**.
- Example of control coupling:
  - A **flag set** in one module and tested in **another module**.

Common Coupling

- Two modules are **common coupled**
  - If they share **some global data**.

Content Coupling

- Content coupling exists between two modules:
  - If they **share code**
  - e.g, **branching** from one module into another module.
- The degree of **coupling increases**
  - from **data coupling to content coupling**.

Design Approaches

- Two fundamentally different software design approaches
  - **Function-oriented design**
  - **Object-oriented design**

Function-Oriented Design

- A system is looked upon as something that performs a **set of functions**.
- Starting at this **high-level view** of the system.
- **Each function** is successively refined into more **detailed functions**.
- Functions are mapped to a **module structure**.

Example

- The function creates a new library member.
  - Creates the record for a **new member**
  - Assigns a **unique membership number**
  - Prints a **bill** for the membership

Example

- Create-library-member function consists of the following sub-functions:
  - **Create-member-record**
  - **Assign-membership-number**
  - **Print-bill**

Function-Oriented Design

- Each **sub-function**:
  - Split into more **detailed sub-functions** and so on.

Function-Oriented Design

- Several **function-oriented design** approaches have been developed in last five decade.
  - Wirth's step-wise refinement (**1971**)
  - Jackson's structured design (Jackson, **1975**)
  - Warnier-Orr methodology (**1976**)
  - **Structured design (1979)**
  - Hatley and Pirbhai's Methodology (**1987**)

Object-Oriented Design

- System is viewed as **a collection of objects** (i.e. entities).
- System state is **decentralized** among the objects:
  - Each object manages its **own state information**.

Object-Oriented Design

- Objects have their **own internal data**:
  - Defines their state.

- **Similar objects constitute** a class.
  - Each object is a member of some class.

- Classes may **inherit features**
  - From a super class.

- Conceptually, objects **communicate** by message passing.

Object-Oriented Design

- **Library Automation Software:**
  - Each library member is a **separate object.**
    - With its **own data and functions**.
  - Functions defined for one object:
    - Cannot directly refer to or change data of other objects.

Object-Oriented versus Function-Oriented Design

- Unlike function-oriented design,
  - In FOD the basic abstraction is functions such as “sort”, “display”, “track”, etc.,
  - Whereas in OOD real-world entities such as “employee”, “picture”, “machine”, “radar system”, etc.

Object-Oriented versus Function-Oriented Design

- In FOD:
  - Software is developed by designing functions such as:
    - update-employee-record,
    - get-employee-address, etc.
  - Whereas in OOD, Software is developed by designing objects such as:
    - employees,
    - departments, etc.

Object-Oriented versus Function-Oriented Design

- **In FOD:**
  - State information is shared in a centralized data.
  - Whereas in OOD the state information is distributed among the objects of the system.

Object-Oriented versus Function-Oriented Design

- The functions are usually associated with specific real-world entities and directly access **only part of the system state information.**

- One object may **discover** the **state information** of another object by **message passing**.

Example:

- In an employee pay-roll system, the following can be global data:
  - employee names,
  - code numbers,
  - basic salaries, etc.

- Whereas, in object oriented design:
  - Data is **distributed among different employee objects** of the system.

Fire-Alarm System

- We need to develop a computerized fire alarm system for a large multi-storied building:
  - There are **80 floors** and **1000 rooms** in the building.

- Different rooms of the building:
  - Fitted with **smoke detectors** and **fire alarms**.

- The fire alarm system would monitor:
  - **Status of the smoke detectors**.

Fire-Alarm System

- Whenever a **fire condition** is reported by any smoke detector:
  - The fire alarm system should:
    - **Determine the location** from which the fire condition was reported
    - **Sound the alarms** in the neighboring locations.
    - **Flash an alarm message** on the computer console:
    - **Fire fighting personnel man** the console round the clock.

Fire-Alarm System

- After a **fire condition** has been successfully handled,
  - The fire alarm system should let fire fighting personnel **reset the alarms.**

Function-Oriented Approach:

- **Global data**
  detector_status
  detector_locs
  alarm-status
  alarm_locs
  neighbor-alarms

  The functions which operate on the system state:
  interrogate_detectors();
  get_detector_location();
  determine_neighbor();
  ring_alarm();
  reset_alarm();
  report_fire_location();

Object-Oriented Approach:

- **class detector**

  attributes: status, location, neighbors
  operations: create, sense-status, get-location, find-neighbors

- **class alarm**
  attributes: location, status
  operations: create, ring-alarm, get_location, reset-alarm

- In the object oriented program,
  - An appropriate number of **instances of the class detector** and **alarm** should be created.

SA/SD (Structured Analysis/Structured Design)

- SA/SD technique draws heavily from the following methodologies:
  - Constantine and Yourdon's methodology
  - Hatley and Pirbhai's methodology
  - Gane and Sarson's methodology
  - DeMarco and Yourdon's methodology
- SA/SD technique can be used to perform
  - **high-level design.**

Overview of SA/SD Methodology

- SA/SD methodology consists of two distinct activities:

  - **Structured Analysis (SA)**
  - **Structured Design (SD)**

- During structured analysis:
  - **functional decomposition** takes place.

- During structured design:
  - **module structure** is formalized.

Structured Analysis

- Transforms a textual problem description into a graphic model.
  - Done using **Data Flow Diagrams (DFDs).**
  - DFDs graphically represent the results of **structured analysis**.

Data Flow Diagrams

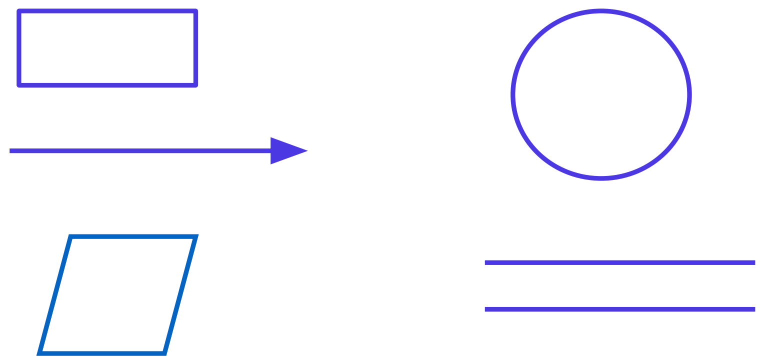
- **DFD is an elegant modelling technique:**
    - Useful not only to represent the results of structured analysis.
    - Applicable to other areas also:
        - e.g. for showing the **flow of documents** or **items in an organization**,
- **DFD technique is very popular:**
    - It is powerful and yet simple to understand and use.

Data Flow Diagrams

- **A DFD model:**
  - Uses limited types of symbols.
  - Simple set of rules
  - Easy to understand:
    - It is a hierarchical model.

Data Flow Diagrams

- Primitive Symbols Used for Constructing DFDs:



External Entity Symbol

- **Represented by a rectangle**
- External entities are **real physical entities**:
  - input data to the system or

  - consume data produced by the system.
  - Sometimes external entities are called terminator, source, or sink.

Function Symbol

- A function such as “search-book” is represented using a **circle**:
  - This symbol is called a **process** or **bubble** or **transform**.
  - Bubbles are annotated with **corresponding function names**.
  - Functions represent **some activity**:

Data Flow Symbol

- **A directed arc or line.**
  - Represents data flow in the **direction of the arrow**.
  - Data flow symbols are annotated with names of data they carry.

Data Store Symbol

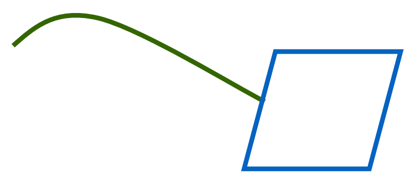
- Represents a logical file:
  - A logical file can be:
    - **a data structure**

    - a **physical file on disk**.

  - Each data store is **connected to a process**:
    - By means of a **data flow symbol**.

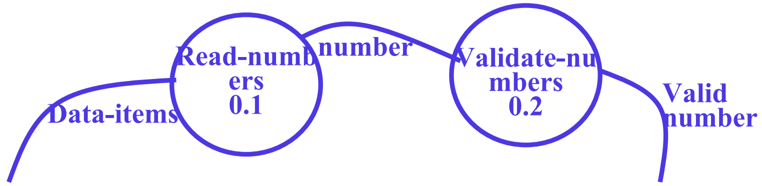
Output Symbol

- Output produced by the system



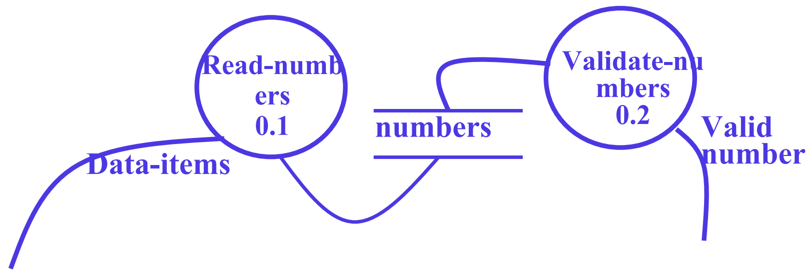
Synchronous Operation

- If two **bubbles are directly connected** by a data flow arrow:
  - They are **synchronous**

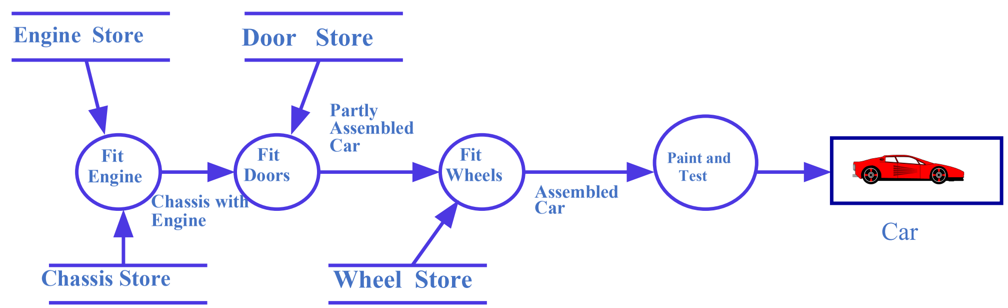


Asynchronous Operation

- If two bubbles are connected via a **data store**:
  - They are **asynchronous**.



Data Flow Model of a Car Assembly Unit



Levels in Data Flow Diagrams (DFD)

- Software engineering DFD can be drawn to represent the system of **different levels of abstraction**.
- **Higher-level DFDs** are partitioned into low-level **DFDs** for more information and functional elements.
- Levels in DFD are numbered 0, 1, 2, and 3.

Level 0 DFD

- This is the **highest-level DFD**, which provides an overview of the entire system.

- It shows the **major processes**, **data flows**, and **data stores** in the system, without providing any details about the **internal workings** of these processes.

- Level O DFD is represented by the **Context Diagram**.

Context Diagram

- A context diagram shows:
  - **Data input** to the system.
  - **Output data** generated by the system.
  - **External entities**.

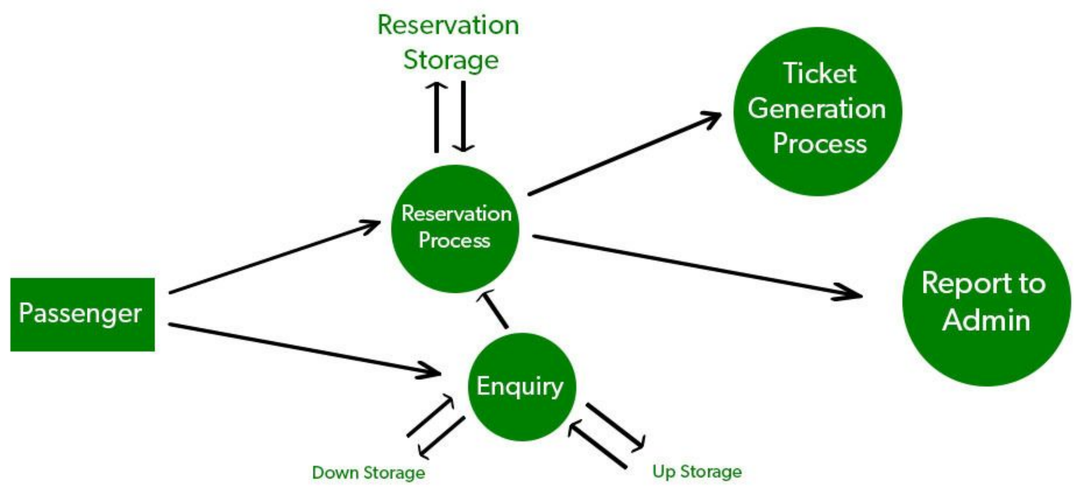
Railway Reservation: Context Diagram

0-LEVEL DFD

Level 1 DFD

- This level provides a **more detailed view of the system** by breaking down the major processes identified in the **level 0 DFD** into sub-processes.
- Each sub-process is depicted as a **separate process** on the level 1 DFD.
- The **data flows and data stores** associated with each sub-process are also shown.

Railway Reservation: Level 1



1-LEVEL DFD

Level 2 DFD

- This level provides an **even more detailed view** of the system by breaking down the **sub-processes identified in the level 1 DFD into further sub-processes**.

- Each sub-process is depicted as a **separate process on the level 2 DFD**.

- The **data flows and data stores** associated with each sub-process are also shown.

Decomposition

- The **decomposition** of a bubble is also called **factoring** or **exploding**.

- Each bubble is decomposed to
  - Between **3 to 7 bubbles**.

- Too few bubbles make decomposition **superfluous**:
  - If a bubble is decomposed into just **one or two bubbles**:
    - Then this decomposition is **redundant.**

- Too many bubbles make the DFD model **hard to understand**
  - More than 7 bubbles at any level of a DFD.

Level 3 DFD

- This is the **most detailed level of DFDs**, which provides a detailed view of the processes, data flows, and data stores in the system.
- This level is typically used **for complex systems**, where a high level of detail is required to understand the system.
- Each process on the level 3 DFD is depicted with a **detailed description** of its input, processing, and output.
- The data flows and data stores associated with each process are also shown

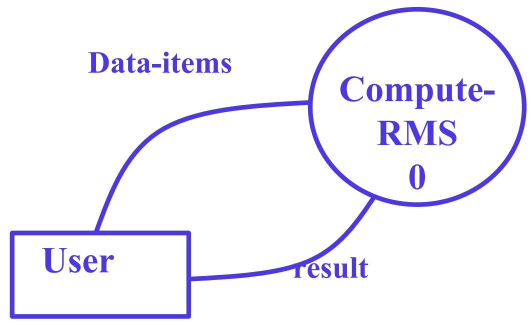
Example: RMS Calculating Software

- Consider a software called RMS calculating software:
  - Reads **three integers** in the range of -1000 to +1000
  - Finds out the **Root Mean Square (RMS)** of the three input numbers
  - **Displays the result**.

Example: RMS Calculating Software

- The context diagram is simple to develop:
  - The system accepts 3 integers from the user.
  - Returns the result to him.

Example: RMS Calculating Software (Level 0 DFD)



Context Diagram

Example: RMS Calculating Software

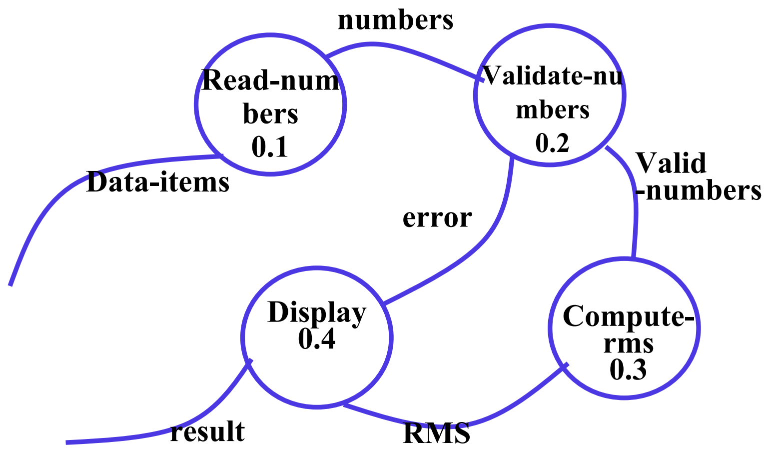
- From a cursory analysis of the problem description:
  - We can see that the system needs to perform several things.

- Accept input numbers from the user:
  - **Validate the numbers**,

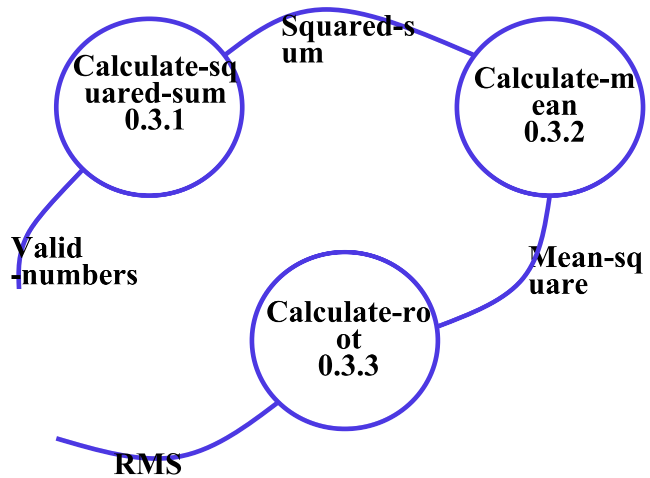
  - **Calculate the root mean square** of the input numbers

  - **Display the result**.

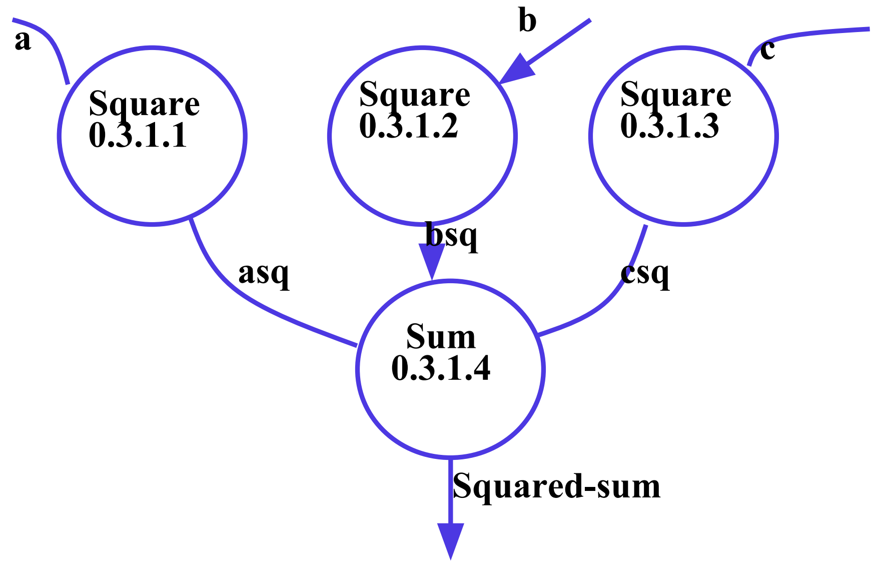
Example: RMS Calculating Software (Level 1 DFD)



Example: RMS Calculating Software (Level 2 DFD)



Example: RMS Calculating Software (Level 3 DFD)



Summary of Structured Analysis

- **Based on principles of:**
  - **Top-down decomposition** approach.

  - **Divide and conquer** principle:
    - Each function is considered **individually** (i.e. isolated from other functions).
    - **Decompose** functions totally disregarding what happens in other functions.

  - **Graphical representation** of results using
    - Data flow diagrams (or bubble charts).

Structured Design

- All the functions represented in the DFD:
  - **Mapped to a module structure**.
- The module structure is also called the **software architecture**:

Data Dictionary

- A DFD is always accompanied by a **data dictionary**.

- A data dictionary lists all **data items appearing** in a DFD:
  - Definition of all **composite data items** in terms of their component data items.

  - All data names along with the purpose of the data items.

- For example, a data dictionary entry may be:

  grossPay = regularPay + overtimePay

Importance of Data Dictionary

- Provides all engineers in a project with **standard terminology** for all data:
  - A consistent vocabulary for data is very important.

  - Different engineers tend to use different terms to refer to the **same data**.
    - Causes unnecessary confusion.

- Data dictionary provides the definition of different data:
  - In terms of their component elements.

- For large systems,

  - The data dictionary grows rapidly in size and complexity.

  - Typical projects can have thousands of data dictionary entries.

  - It is extremely difficult to maintain such a dictionary manually.

End of Unit 4

THANKS…..